Sharing the Architectural Knowledge of Quantitative Analysis

Jansen, Anton; Vries, Tjaard de; Avgeriou, Paris; Veelen, Martijn van

Published in:
QUALITY OF SOFTWARE ARCHITECTURES, PROCEEDINGS

Publication date:
2008

Sharing the Architectural Knowledge of Quantitative Analysis

Anton Jansen[1], Tjaard de Vries[1], Paris Avgeriou[1], and Martijn van Veelen[2]

[1] University of Groningen, Department of Mathematics and Computing Science,
P.O. Box 800, 9700AV Groningen, The Netherlands
anton@cs.rug.nl, tjaard@tjaard.nl, paris@cs.rug.nl
[2] ASML, DE-SSD, Litho Systems Architecture
5504DR Veldhoven, The Netherlands
martijn.van.veelen@asml.com
Previously employed by ASTRON
P.O. Box 2, 7990AA Dwingeloo, The Netherlands

Abstract. Sharing the architectural knowledge of architectural analysis among stakeholders proves to be troublesome. This causes problems in and with architectural analysis, which can have serious consequences for the quality of a system being developed, as this quality might be incompletely or wrongly assessed. This paper presents a domain model, which can be used as a common ground among analysts and architects to capture and explicitly share such knowledge. This enables a way to overcome some of the obstacles imposed by the multi-disciplinary context in which architectural analysis takes place. To apply the domain model in practice, we have created a tool implementing (part of) this domain model for capturing and using explicit architectural knowledge during analysis. We validate the tool and domain model in the context of an industrial case study.

1 Introduction

Expectations, and therefore demands, on the quality of systems are ever increasing. More and more systems become software-intensive systems, in which software plays a crucial role in the delivery of the required functionality. Consequently, the quality of these systems is greatly influenced by the quality of the software. Software architectures offer the ability to predict the expected quality of a software system before it is actually implemented or changed [1]. This architectural analysis gives engineers a tool to find out which kind of software system is optimal for their system needs, without implementing the (changes to the) software beforehand.

Architectural Knowledge (AK) [2,3,4] plays a crucial role in architectural analysis, as it is this knowledge an analyst consumes and produces [5]. System analysts are often experts in certain domains and use or *consume* AK to analyze (parts of) an architecture. During the analysis, AK is *produced* by analysts, which ranges from individual analysis results to new insights into the overall behavior of the design space.

S. Becker, F. Plasil, and R. Reussner (Eds.): QoSA 2008, LNCS 5281, pp. 220–234, 2008.

However, sharing an architectural analysis among system analysts and other relevant stakeholders proves to be problematic. To fully understand the results of an architectural analysis, the AK *consumed* to produce these documents and models is required as well, since this AK explains the reasoning path the analyst used to come up with the analysis result. Often not all of this knowledge is shared. This incomplete AK sharing can have repercussions for the quality of the analysis, thus negatively affecting the quality of the realized or changed system. Typically, these repercussions include: (1) difficult communication among stakeholders/analysts, (2) troublesome integration of analysis results of different analysts and (3) incomplete documentation and traceability of the analysis itself.

The incomplete AK sharing has two major causes. First, awareness is often missing of which AK is relevant to share. Second, the multi-disciplinary context, i.e. the very different backgrounds of the stakeholders and analysts, creates an obstacle to sharing this knowledge.

In this paper, the focus is on sharing the AK of quantitative analysis. First, the needs are identified for sharing this AK. Based on this, the AK relevant to share is identified and described in a domain model. The domain model describes this knowledge independently of the background of the stakeholders and analysts, thus creating a common ground, which to some extent can prevent issues arising from the multi-disciplinary context.

The rest of this paper is organized as follows. Section 2 presents the needs to share the AK of quantitative analysis. Section 3 presents a domain model describing the AK of quantitative analysis relevant to share. This is followed by section 4, which presents a tool for sharing this knowledge for one of the identified needs. The domain model and tool are validated with a case study in section 5. Related work is discussed in section 6 and the paper concludes with conclusions and future work in section 7.

2 Sharing the AK of Quantitative Analysis

In software architecting, the evaluation of software architectures forms an important activity [6]. Based on evaluation results, informed architectural decisions can be made. Not only does this increase the confidence in the architectural design, but also makes the design easier to defend, as objective rationale and alternatives for the architectural decisions are available. One way to obtain such results is by quantitative analysis. In quantitative analysis, one or more analysis models are created to quantitatively analyze various potential architectural solutions. The results of an analysis model are quantifications of one or more quality attributes.

Sharing the AK of quantitative analysis is required for the following three cases:

- **Integration.** For complex systems, a divide and conquer strategy is often used for quantitative analysis. The system is divided into subsystems, for each of which analysis is performed by analysts, each with their own area of expertise. Another way to divide the system is based on the relevant quality

attributes, each of which is to be analyzed by its corresponding experts. Consequently, numerous analysis models are created.

However, to evaluate a system as a whole, the different perspectives the analysts have on the architecture need to be combined. Hence, *integration* of their analysis models is required. This requires the analysis models to be compatible with each other, i.e. use the same kind of quantification and terminology for the various quality attributes. In addition, assumptions made about the solutions being evaluated in the analysis models should be synchronized to ensure that all analysis share a common ground. Thus, considerable knowledge sharing of the consumed and produced AK is required among the analysts.

- **Verification.** Another need for AK sharing comes from *verification*, i.e. the need to verify the correctness, completeness, and consistency of the analysis models. Typically, analysts let their models be reviewed by others, as to find problems with their analysis. This is especially useful if the analysis is performed on the boundary of the expertise of different analysts.
- **Validation.** A third need for AK sharing comes from stakeholders who want (or need) to *validate* a design, i.e. want to have a clearer understanding of the rationale used for an architectural decision. This knowledge can be used to convince stakeholders of the appropriateness of an architectural decision, e.g. in the form of traceability. Additionally, this can create further insight, allowing the discovery of new and potentially better alternative designs.

3 Domain Model for Quantitative Analysis

3.1 Introduction

Before the AK of quantitative analysis can be shared, we need to know what this knowledge entails. To discover this knowledge, we have investigated architectural analysis in a particular organization: Astron, the Dutch national foundation for research in Astronomy. One of their activities is performing quantitative architectural analysis of radio and optical telescopes. In the new generation of telescopes, software has become a dominant design factor. Astron analysts mainly perform quantitative performance and cost analysis (see e.g. [7]), although qualities like reliability and maintainability are quantitatively analyzed sometimes as well.

We closely cooperated with Astron analysts to find out which AK is consumed and produced during their analysis. To describe this knowledge, we have developed a domain model. The model describes the concepts and the relationships of this knowledge. It is based on informal interviews, inspection of various analysis models, software architecture documents and system analysts meetings. In addition, we were inspired by some of the concepts and insights gained from the Massive project, which delivered a tool for quantitative cost and performance analysis of embedded systems of telescopes [8]. Furthermore, the domain model has been improved in multiple iterations with the system analysts to make sure that it reflects their practice.

The aim was to come up with a unified domain model, which was independent of the individual analysts, their quantitative modeling approaches, and the qualities being analyzed, since only such a domain model would become effective in tackling the multi-disciplinary boundary by providing a common ground. Although the naming of many concepts in the domain model are specific to Astron, most of the concepts will be likely found back under different names in other organizations as well. To which extent this is the case is still an open research question.

In the remainder of this section, we present this domain for quantitative analysis. Due to the many concepts and their relationships, the model is presented in parts to enhance readability. For the full details of the domain model and a complete figure, we refer to [9].

3.2 AK Basis Model

The AK basis model presented in figure 1 is not part of the domain model for quantitative analysis. Rather, it presents the basic concepts that individual domain models extend for their particular case. In this case, it forms the foundation upon which the domain model for quantitative analysis is built. At the heart of the model is the concept of a *Knowledge Entity*: a concept, which represents the different knowledge entities found in a particular domain. For example, a requirement is a *Knowledge Entity* in a domain model for architectural design documentation. In a domain model, *all* the domain concepts inherit from *Knowledge Entity*. A domain model can define specific relationships among *Knowledge Entities*, thereby relating them to each other. How and what the semantics of these relationships are is not known to the AK basis model.

Knowledge Entities typically have one or more creators called *Authors*, who express a *Knowledge Entity* in one or more *Artifact Fragment*s. An *Artifact Fragment* identifies which part of an *Artifact* contains a (partial) description of a *Knowledge Entity*. For example, a paragraph (i.e. an *Artifact Fragment*) describing a specific stakeholder (i.e. a *Knowledge Entity*) is contained in a Word document (i.e. an *Artifact*).

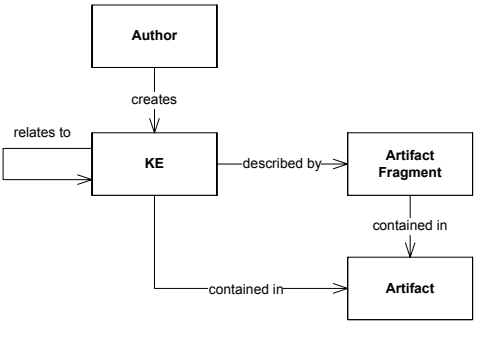

Fig. 1. The AK basis model

3.3 Quantitative Analysis Process

The domain model part that models the concepts of the quantitative analysis process is presented in figure 2. The goal of quantitative analysis is to investigate the quality of different design options, or *Alternative*s. At Astron, *Alternative*s are stratified. The first classification is in *Design Concept*s; the basic type of design. A *Design Concept* outlines a basic solution direction, thus defining a scope in the (large) design space. For radio telescopes, examples include: single dish (i.e. make one big dish), phased arrays (i.e. combine multiple antennas using beamforming), and aperture arrays (i.e. correlate the signals of telescopes and phased arrays). A *Design Concept* is specialized in the analysis in one or more *Scenario*s, which are quantitatively analyzed in one or more *Analysis Model*s.

An *Analysis model* consists of *System Parameter*s, which are the input and output of *Analysis Function*s. A *System parameter* describes the state of part of the *Analysis Model*, which is expressed in a *Number* of a unit defined by the *System Parameter*. An *Analysis Function* describes a *System Parameter*'s behavior and relationships. For example, the cost of a dish is a *System Parameter*, which can be the output of an *Analysis Function* that takes as inputs the *System Parameter*s of the costs of the various parts of a dish. Some of the *System Parameter*s can be related to one or more *Quality Attribute*s, which are defined by the quality model(s) used in the analysis. This relationship is used to classify the *System Parameter*s based on the quality they contribute to.

An *Analysis Model* is an aggregation of *System Parameters, Analysis Functions, Quality Attributes* and *Numbers*. The input of an *Analysis Model* is set by

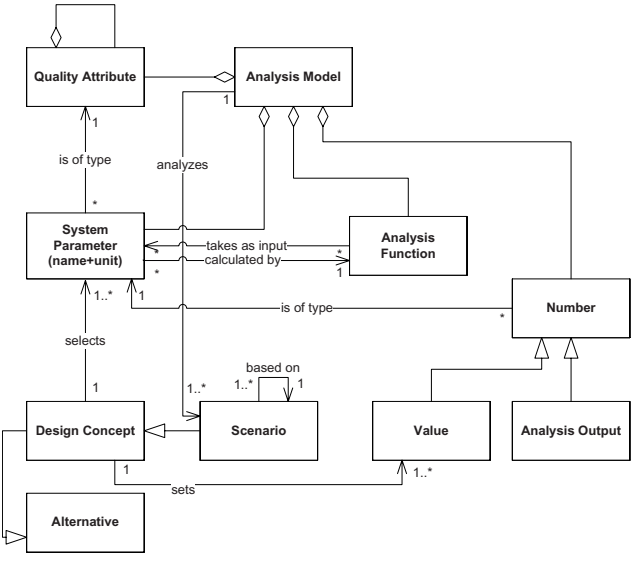

Fig. 2. Quantitative Analysis Process Concepts

assigning *Values* to input *System Parameter*s. The output of an *Analysis Model* are *Analysis Output*s, which are *Number*s of *System parameter*s calculated by *Analysis Function*s. In the example of the dish costs, the costs in dollars of the dish is an output *System Parameter* associated with the *Quality Attribute* costs and their value (i.e. a *Number*) is an *Analysis Output*. In a similar fashion, the parts of a dish are modelled as input *System parameter*s and *Value*s. Please remark that a distinction between input and output *System Parameter*s is not explicit in the model, but rather is inferred from the relationships they have with the *Analysis Function*s.

*Analysis function*s have an additional property, which is not visualized in figure 2. *Analysis Model*s are typically incomplete, as analysts will utilize shortcuts in their models based on their (tacit) domain knowledge and intuition. Hence to address this issue, the domain model defines a *Confidence* property for an *Analysis Function*. This property indicates to which extent the *Analysis Function* is reliable and whether the *Analysis Function* is based on intuition, trends, fact, or simply a wild guess. Consequently, points for improvements in an *Analysis Model* are made explicit.

3.4 Integration of Analysis Models

The previous subsection explained the concepts involved in quantitative analysis for a single analysis model. However, as identified in section 2 there often is a need to share individual analysis models, due to the divide and conquer strategy being used for analysis. Only when the analysis models of the parts are unified (and therefore shared), a complete architectural evaluation of an *Alternative* becomes feasible.

Figure 3 presents the concepts involved with the integration of *Analysis model*s. First of all, the figure visualizes that each *Analysis Model* has its own *Namespace*. It is in bridging these *Namespace*s of various *Analysis Model*s that the sharing

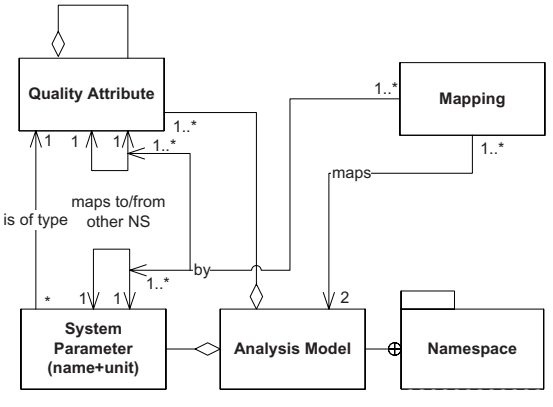

Fig. 3. Concepts for integrating Analysis Models

of AK takes place among different analysts. To integrate two analysis models, a *Mapping* should be defined between them. This involves defining mapping relationships between the *Quality Attribute*s (i.e. the quality model) and the *System Parameter*s of the two *Analysis Model*s.

One effect of these *Mapping*s is that it makes the dependencies among *Analysis Model*s visible. For example, one could identify which output *System Parameter*s of one *Analysis Model* are used in other *Analysis Models*. However, more important is the fact that an analyst can follow the translation made from his/her *System Parameter*s and *Quality Attribute*s terms into the terms used in an *Analysis Model* created by a colleague. For example, the cost of a dish could be called costs in one model and dish cost in another model. Mappings between these two *System Parameter*s identify that both denote the same concept.

Another effect of these *Mapping*s is that the overlap and gaps between *Analysis Model*s are identified. This helps analysts identify semantic inconsistencies among the *Quality Attribute*s and the *System Parameter*s of different *Analysis Model*s. For example, the *System Parameter* costs of a dish could be expressed in one *Analysis Model* in terms of millions of dollars, whereas another *Analysis Model* assumes these costs to be in thousands of dollars. *Mapping*s or their absence, makes such inconsistencies visible.

3.5 Verification of Analysis Models

Another important form of AK sharing during architectural analysis is for the purpose of verification, which is achieved in Astron by reviewing. In reviews, analysts look for problems with the *Analysis Model*s. The feedback gathered from reviews can improve the analysis quality and thus (hopefully) improve the quality of the resulting system. Figure 4 presents the concepts involved in reviewing an *Analysis Model*.

At the heart is the concept of a *Reviewable*, which denotes a concept that is relevant during reviewing. A *Reviewable* has a *Review State*, which is determined by the judgment of one or more *Reviewers*. Concepts that are *Reviewable* include: *Analysis Function*, *Mapping*, *System Parameter*, and *Components*. *Components* are a special type of *System Parameter* useful during reviewing. They provide a mechanism to group large numbers of *System Parameter*s together in a hierarchical fashion (i.e. *Components* can contain other *Components*). Different composition strategies can be used (e.g. process or deployment) to group *System Parameter*s together to create a *View/Topology* on the analysis.

The concepts of *Component* and *View/Topology* allow a *Reviewer* to judge a group of *System Parameter*s as a whole, thereby providing an explicit mechanism to review at different levels of abstraction.

3.6 Validation of Designs Using Analysis Models

Quantitative analysis is not done without a reason. The outcome plays an important role in the architectural design process. This process can be seen as a decision making process, in which an architectural decision has to be made among

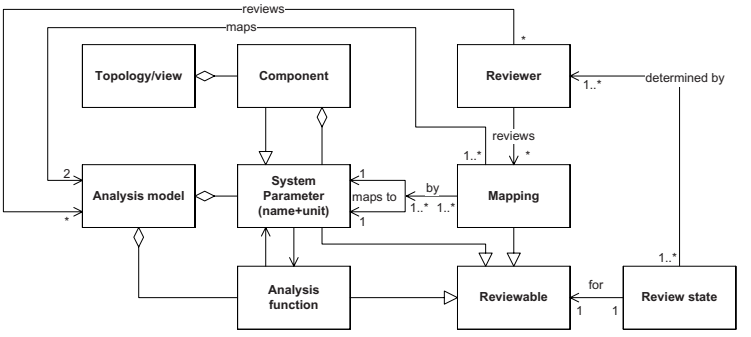

Fig. 4. Concepts of Reviewing and Quantitative Analysis

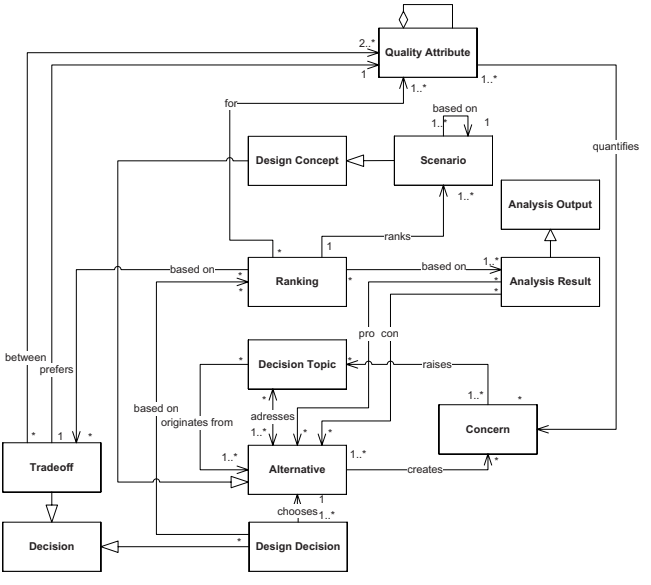

Fig. 5. The Quantitative Analysis and Design Decision process

different *Alternatives* [2,10]. The concepts involved with this decision making process and the role quantitative analysis concepts play in it are visualized in figure 5.

At the core of the decision making process are the concepts of *Decision Topic*s, *Alternative*s, and *Concern*s. *Concerns* (such as requirements) raise *Decision Topic*s to deal with them. The *Decision Topic*s originate from various alternatives that could address the *Concern*. *Alternatives* themselves usually come with side effects, which cause new *Concern*s. When a decision is made for a certain *Alternative*, this is called a *Design Decision*.

When performing quantitative analysis, this *Design Decision* is made on the basis of quantitative knowledge. More precisely, it is based on a quantitative *Ranking* of *Scenario*s with respect to one or more *Quality Attribute*s. The values associated with these *Quality Attribute*s stem from the *Analysis Result*s of the *Scenario*s.

It is not uncommon for quality attributes to be in conflict in the context of a *Ranking*. To come to a *Design Decision*, a *Tradeoff* is made between them. The *Tradeoff* expresses a preference for a certain *Quality Attribute* for a particular *Ranking*. How these tradeoffs are exactly done can vary greatly from case to case and the design method being used.

4 The Knowledge Architect Tool Suite

Astron analysts create analysis models using both general purpose and domain specific tools. General purpose tools include Matlab, Python, Microsoft Excel, white boards and pieces of paper. An example of a domain specific tool is the Massive tool [8] for cost and performance analysis of embedded systems.

The domain model of section 3 should describe all the AK consumed and produced in these tools. To validate the domain model with respect to *verification* (see section 2), we have created the Knowledge Architect Excel plug-in tool [11], which implements the domain model for one of these general purpose tools (i.e. Microsoft Excel). The tool supports analysts in making the AK produced during analysis explicit. The aim is to facilitate the sharing of AK for verification of Excel analysis models by other analysts. The other AK sharing purposes (i.e. integration and validation) are supported by other tools, which are part of the larger Knowledge Architect tool suite.

In Excel, *System Parameter*s, *Analysis Function*s, and *Number*s have a strong relationship, as these three concepts are joint together in the form of a cell. The visible representation of the cell is normally the *Number* of a *System Parameter*. The equation bar presents the *Analysis Function* of a *System Parameter* if the appropriate cell is selected. By design, Excel does not allow for separation between these concepts. Thus the only way to have multiple *Scenario*s share the same *Analysis Function* for a *System Parameter* is by duplicating a cell.

Often labels surrounding the cell denote the semantic meaning of a cell (and thereby of a *System Parameter* and its associated *Number*). However, the texts of these labels are not formally related to any cells or *System Parameter*s. The tool allows analysts to make special annotations to make these relationships explicit. Figure 6 presents how a cell is annotated. For a *System Parameter* of a cell, a name, symbol, unit, and description can be defined. In the same window, an analyst can define the confidence of the *Analysis Function* as well. The *Author* of these *Knowledge Entities* is automatically determined.

The window presented in figure 6 does not show all of the KE types covered by the tool. A similar window exists for relating *System parameter*s to *Scenario*s. Another window allows different *Reviewer*s to define the *Review State* of a *System Parameter* and *Analysis Function* and give comments on them.

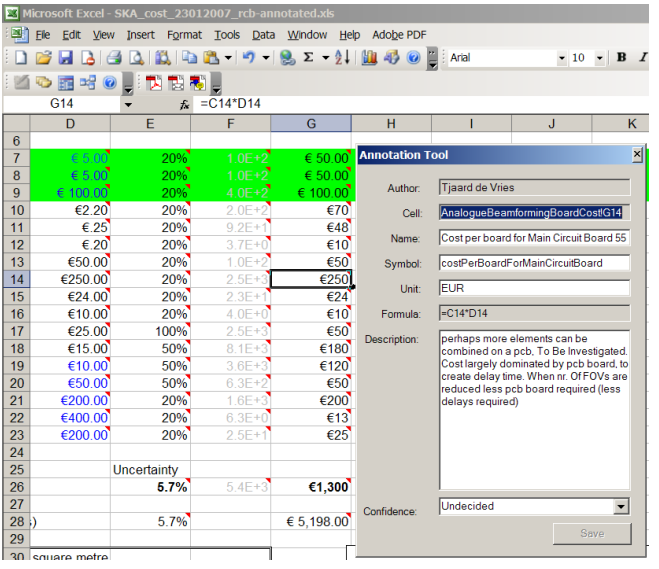

Fig. 6. Knowledge Architect Excel Plugin annotating AK in Excel

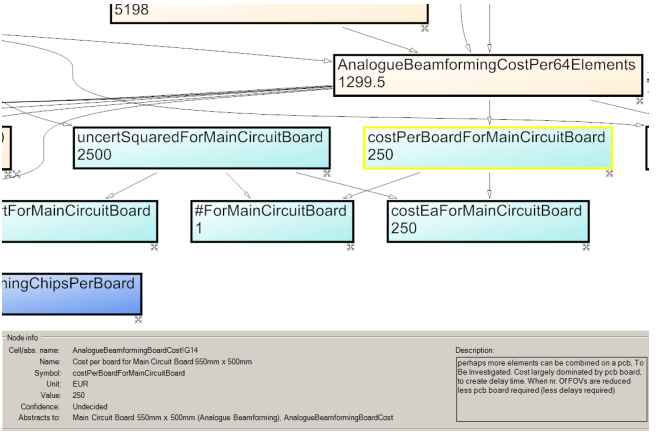

Fig. 7. An excerpt of the system parameter dependency graph

Furthermore, since annotating cells one by one is time-consuming, a feature is available to annotate whole tables of cells without much effort.

To facilitate verification, the tool offers a visualization of the dependency graph. An example of this graph is presented in figure 7. A node in the graph represents a *System Parameter* or a *Component* , i.e. a set of *System Parameter*s. An arrow between two nodes indicates that the *Analysis Function*(s) of the *System Parameter*(s) of one node uses the *System Parameter*s in its calculations,

thereby creating a dependency between the *System Parameter*s. The name of a node comes either from a user annotation or a default cell name. The bottom part of the figure visualizes the details of a selected node.

There is a correspondence between the nodes in the dependency graph and the cells of the Excel worksheets. Making a selection of cells in a worksheet, also selects the related nodes in the dependency graph and vice versa. An analyst can use this selection mechanism to easily create new *Component*s. The color of a node in figure 7 indicate the *Component* a *System Parameter* belongs to. A user can expand or collapse nodes, thereby providing a way to view either more abstract *System Parameter*s or more detailed ones. Orthogonal to this, is the ability to filter *System Parameter*s based on the *Scenario*s they are involved in.

5 Experiment: Sharing a SKA Cost Model

5.1 Introduction

In this section we present an experiment to find out to what extent the Knowledge Architect Excel plug-in helps in sharing AK for verification purposes. Especially, we want to know whether an analyst understands someone else's analysis model better and more quickly when aided by the tool. Furthermore, we want to know the reasons for any differences found.

The experiment takes place in the context of sharing a cost model of the Square Kilometre Array (SKA) [12]. SKA is a world wide international scientific instrument, which is still in its design phase and will consist of a radio signal collectable area of one square kilometre. In cooperation with the Jodrell Bank Centre for Astrophysics in Manchester, UK, Astron has devised a cost model for SKA in Excel. The cost model is rather complex and consists of over 1500 different *System Parameter*s spread over 12 different Excel worksheets. The various designs evaluated have costs in the order of billions (10^9) of dollars.

In this experiment, five Astron analysts were the subjects among which this cost model was shared for verification. First, the analysts were given a training to familiarize themselves with the tool. After this they were given tasks in the form of time-boxed questions, one set to be completed with and one to be completed without the tool. Besides timing the participants, we observed and debriefed the analysts to gather additional qualitative data. The questions used are representative of questions analysts may ask during the verification of analysis models. The questions are divided into the following three classes that denote activities done during verification: (1) **Sensitivity analysis**. This type of task requires an analyst to investigate the dependencies among *System parameter*s to verify their correct interaction. (2) **Consistency analysis**. This type of task requires an analyst to verify whether the same kind of *Analysis Function*s are used for similar *System Parameter*s for different *Scenario*s. (3) **Defect analysis**. This type of task requires an analyst to spot errors in the *Analysis Function*s, their relationships and the assumed *System Parameter*s for a *Scenario*.

In the remainder of this section, we present the lessons learned based on our qualitative results. For more information about the experiment, we refer to [9].

5.2 Lessons Learned

The experiment confirmed that verifying complex analysis models in detail is not an easy task. The analysts mentioned having difficulties with understanding the used terminology inside the cost model. Although the tool did help the subjects to understand the concepts used, it failed to overcome differences in terminology used for the names and descriptions of the *System Parameter*s. Integrating of analysis models, as described in section 3.4, might help in addressing this issue. Since such an integration creates a deeper understanding among the analysts of the differences in terminology they use within their models.

The navigation capabilities of the dependency graph were found useful to find relevant parts of the analysis model. However, for understanding the analysis itself, the vast multitude of arrows and nodes in the graph was too confusing. Surprisingly enough, this did not matter to the co-author of the cost model; he could directly spot some defects simply by looking at the dependency graph. This indicates that he has some tacit knowledge for filtering out irrelevant information so that he was able to use the tool to quickly get new insights into his own work.

It appeared that the cost model heavily relied on tables. For two-dimensional data, tables in a spreadsheet are a pretty good representation, but for more complex relations between system parameters the tool had a clear edge over plain Excel. It made non-trivial relationships explicit, clearly visible and thus easier to inspect for the subjects than in a spreadsheet.

Overall, it appears that the tool has potential to give extra insight into analysis models, but in its current form fails to deliver in all cases. The domain specific naming and description of the *System Parameter*s is one cause for this. Another cause is that the system parameter graph is too complex, i.e. it fails to reduce the complexity of the analysis model. Improvements in the visualization and the way non-relevant information is filtered out should help with addressing this issue. On a positive side, the tool did help analysts quickly locate relevant parts during verification and authors of a domain model with locating defects and inconsistencies.

6 Related Work

For software architecture evaluation, two types of approaches can be discerned [13]: scenario-based [14], and quantitative model based methods. In scenario-based methods (e.g. ATAM [15], ALMA [16], SALUTA[17], PASA [18] (performance), scenarios are used to define use-cases of the typical and expected future uses of the system. Based on these scenarios one or more architectural designs are evaluated for the qualities of interest. ATAM [15] is a framework method that incorporates the results of other scenario based evaluation methods and focusses on making tradeoffs between different qualities. From a knowledge perspective, this is the decision part of the domain model.

ALMA [16] provides an analysis method for modifiability, likewise SALUTA [17] does this for usability. PASA [18] combines a scenario-based approach with a

quantitative model for performance. PASA requires quantitative goals, performs the analysis quantitatively where possible, and includes a cost-benefit analysis as one of its steps.

A general approach to quantitative architectural analysis for multiple quality attributes is proposed by Bachmann et al. in the form of so-called *reasoning frameworks* [19]. A reasoning framework is a quantitative analysis model with respect to a certain quality attribute. It proposes architectural tactics to improve the architecture with respect to this quality attribute. In essence, a reasoning framework is a quantitative analysis model. Hence, our domain model should describe the AK concepts used in them.

Massive [8,20] is an analytic Layered Queueing Network method aimed at the design of large and complex embedded systems, which focusses on quantitative cost and performance analysis. The approach emphasizes reuse of components and the topology of the system being designed. Our domain model abstracts some of the core concepts of this approach and presents a more precise identification of the concepts involved.

Compared to other AK meta-models and tools, e.g. the Core model [2], AREL [21], Pakme [22], ADDSS[23], the domain model has an extensive description of quantitative AK. The other meta-models exclusively focus on qualitative rationale with no to little attention on how this rationale relates to quantitative AK.

Farenhorst et al. [24] identify difficulties that arise when sharing AK, as well as prerequisites for sharing to be successful. They define incentives for people needed to be willing to share AK, as well as the lesson that striving for completeness is infeasible. However, they do not define the actual knowledge to be shared, as is done in our domain model.

7 Conclusions and Future Work

This paper identified three different needs (i.e. integration, verification, and validation) for sharing the AK of quantitative analysis. For each need, the presented domain model describes the concepts and relationships of the relevant AK. Based on this domain model, a tool was created to facilitate AK sharing for verification. In an experiment, this tool and the domain model were tested.

Based on the lessons learned in this experiment, we can conclude that there is a close relationship between the knowledge needed for verification and integration. Although the domain model provides a common language for the concepts, it does not provide a common language for the instances. They are still domain specific, thus synchronization at this level is still needed for both integration and verification. To what extent this is also the case for validation is an open question for future work.

Another open question has to do with the perceived complexity of an analysis model. An interesting starting point for this is an observation made during the experiment. The co-author of the analysis model used was capable of dealing with this complexity, whereas others were not. The question is which tacit knowledge was involved in this.

How well the presented domain model is generally applicable is an open question as well. For this, we plan to investigate how concepts found in another organization map to ones in the presented domain model. This will provide us with insight into the extent to which the domain model is generally applicable.

In future work, we plan to improve our tool based on the outcomes of these questions. In addition, we plan to create tools, as part of the Knowledge Architect platform, for sharing AK for validation and integration purposes.

Acknowledgements

This research is sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge. We thank the people from Astron who participated in this research. In particular, Kjeld van der Schaaf and Albert-Jan Boonstra.

References

1. Bass, L., Clements, P., Kazman, R.: Software architecture in practice, 2nd edn. Addison-Wesley, Reading (2003)
2. de Boer, R.C., Farenhorst, R., Lago, P., van Vliet, H., Jansen, A.G.J.: Architectural knowledge: Getting to the core. In: Overhage, S., Szyperski, C.A., Reussner, R., Stafford, J.A. (eds.) QoSA 2007. LNCS, vol. 4880. pp. 197–214. Springer, Heidelberg (2008)
3. Kruchten, P., Lago, P., van Vliet, H.: Building up and reasoning about architectural knowledge. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214. Springer, Heidelberg (2006)
4. Habli, I., Kelly, T.: Capturing and replaying architectural knowledge through derivational analogy. In: SHARK-ADI 2007: Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, Washington, DC, USA, p. 4. IEEE Computer Society, Los Alamitos (2007)
5. Lago, P., Avgeriou, P.: First workshop on sharing and reusing architectural knowledge. SIGSOFT Software Engineering Notes 31(5), 32–36 (2006)
6. Clements, P., Rick Kazman, M.K.: Evaluating Software Architectures - Methods and Case Studies. The SEI Series in Software Engineering. Addison-Wesley, Reading (2002)
7. Alliot, S.: A performance cost estimation model for large scale array signal processing system specification. In: Proc. of the Third International Samos Workshop on Synthesis, Architectures, and Simulation, pp. 156–160 (July 2003)
8. Alliot, S., Nicolae, L., van Veelen., M.: A tool for exploring the large scale signal processing systems specifications. In: IEEE International conference on parallel computing in electrical engineering, pp. 341–348 (September 2004)
9. de Vries, T., Jansen, A.G.J.: Knowledge architect excel plug-in technical report. Technical Report IWI preprint 2008-7-01, Department of Mathematics and Computing Science, University of Groningen, PO Box 800, 9700 AV The Netherlands (March 2008)

10. Jansen, A.G.J., Bosch, J., Avergiou, P.: Documenting after the fact: recovering architectural design decisions. Journal of Systems and Software 81(4), 536–557 (2008)
11. The Griffin project website, http://search.cs.rug.nl/Griffin
12. The Square Kilometre Array project website, http://www.skatelescope.org/
13. Babar, M.A., Gorton, I.: Comparison of scenario-based software architecture evaluation methods. In: Software Engineering Conference, 2004. 11th Asia-Pacific, 30 November-3 December, pp. 600–607 (2004)
14. Dobrica, L., Niemela, E.: A survey on software architecture analysis methods. IEEE Trans. Softw. Eng. 28(7), 638–653 (2002)
15. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures, Views and Beyond. Addison-Wesley, Reading (2002)
16. Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H.: Architecture-level modifiability analysis (ALMA). J. Syst. Softw. 69(1-2), 129–147 (2004)
17. Folmer, E., van Gurp, J., Bosch, J.: Software architecture analysis of usability. In: 9th IFIP Working Conference on Engineering for Human-Computer Interaction, pp. 321–339 (July 2004)
18. Williams, L.G., Smith, C.U.: Pasa: a method for the performance assessment of software architectures. In: WOSP 2002: Proceedings of the 3rd international workshop on Software and performance, Rome, Italy, pp. 179–189. ACM Press, New York (2002)
19. Bachmann, F., Bass, L., Klein, M., Shelton, C.: Designing software architectures to achieve quality attribute requirements. IEE Proceedings - Software 152(4), 153–165 (2005)
20. Alliot, S., M.: Modelling and system design for the lofar station digital processing. In: SPIE Astronomical Telescopes and Instrumentation, Modelling and System Engineering (June 2004)
21. Tang, A., Jin, Y., Han, J.: A rationale-based architecture model for design traceability and reasoning. Journal of Systems and Software 80(6), 918–934 (2007)
22. Babar, M.A., Gorton, I., Kitchenham, B.: A framework for supporting architecture knowledge and rationale management. In: Dutoit, A.H., McCall, R., Mistrík, I., Paech, B. (eds.) Rationale Management in Software Engineering, pp. 237–254. Springer, Heidelberg (2006)
23. Capilla, R., Nava, F., Pérez, S., Dueñas, J.C.: A web-based tool for managing architectural design decisions. SIGSOFT Software Engineering Notes 31(5) (2006)
24. Farenhorst, R., Lago, P., van Vliet, H.: Prerequisites for successful architectural knowledge sharing. In: ASWEC 2007: Proceedings of the 2007 Australian Software Engineering Conference, pp. 27–38. IEEE Computer Society, Los Alamitos (2007)